Empirical Software Engineering manuscript No.
(will be inserted by the editor)

An Empirical Study on the Issue Reports with Questions Raised during the Issue Resolving Process

Yonghui Huang · Daniel Alencar da Costa · Feng Zhang · Ying Zou

Author pre-print copy. The final publication is available at Springer via:
http://dx.doi.org/10.1007/s10664-018-9636-3

Abstract An issue report describes a bug or a feature request for a software system. When resolving an issue report, developers may discuss with other developers and/or the reporter to clarify and resolve the reported issue. During this process, questions can be raised by developers in issue reports. Having unnecessary questions raised may impair the efficiency to resolve the reported issues, since developers may have to wait a considerable amount of time before receiving the answers to their questions. In this paper, we perform an empirical study on the questions raised in the issue resolution process to understand the further delay caused by these questions. Our goal is to gain insights on the factors that may trigger questions in issue reports. We build prediction models to capture such issue reports when they are submitted. Our results indicate that it is feasible to give developers an early warning as to whether questions will be raised in an issue report at the issue report filling time. We examine the raised questions in 154,493 issue reports of three large-scale systems (i.e., Linux, Firefox and Eclipse). First, we explore the topics of the raised questions. Then, we investigate four characteristics of issue reports with raised questions: (i) resolving time, (ii) number of developers, (iii) comments, and (iv) reassignments. Finally, we build a prediction model to predict if questions are likely to be raised by a developer in an issue report. We apply the random forest, logistic regression and Naïve Bayes models to predict the possibility of raising questions in issue reports. Our prediction models obtain an Area Under Curve (AUC) value of 0.78, 0.65, and 0.70 in the Linux, Firefox, and Eclipse systems, respectively. The most important variables according to

Y. Huang, D.A. da Costa and Y. Zou
Department of Electrical and Computer Engineering, Queen's University,
Kingston, Ontario, Canada
E-mail: {ckeys.huang, daniel.alencar, ying.zou}@cs.queensu.ca

F. Zhang
School of Computing, Queen's University, Kingston, Ontario, Canada
E-mail: {feng}@cs.queensu.ca

our prediction models are the number of Carbon Copies (CC), the issue severity and priority, and the reputation of the issue reporter.

Keywords issue reports; questions; bug fixing; empirical study

1 Introduction

The delay in resolving issues can affect the satisfaction of users (Breu et al., 2009). Investigating the factors that impact the efficiency of the issue resolving process has attracted both academia and industry (Baysal et al., 2009; da Costa et al., 2017a,b; Francalanci and Merlo, 2008; Ghapanchi and Aurum, 2011; Nguyen et al., 2012; Ohira et al., 2012). An important factor that affects the efficiency of resolving process is the interaction among developers and reporters to discuss how to address a reported issue during the issue resolving process. Breu et al. (Breu et al., 2010) observe that the active participation of both reporters and developers in the discussion of an issue reports may increase the efficiency of resolving issues. When resolving issues, developers might raise questions in order to clarify the described issue (Breu et al., 2010; Ko and Chilana, 2011; Lotufo et al., 2012) or the requested feature (Ko and Chilana, 2011). As a consequence, we are interested in performing a more detailed empirical study on a larger scale of issue reports. Our goal is to gain insights about the impact of raised questions in issue reports on the issue resolution process.

Much research has been devoted to study the discussion between developers and reporters in issue reports. A preliminary study by Breu et al. (Breu et al., 2009) analyzes 600 issue reports to study how the raised questions in issue reports are answered. Sillito et al. (Sillito et al., 2006) investigate what information developers generally need in issue reports and how developers usually obtain such information. Likewise, Bettenburg et al. (Bettenburg et al., 2008a) surveyed 466 developers and reporters to understand the characteristics of a well-written issue report. The authors also propose CUEZILLA, a recommender tool that indicates how issue reports can be improved.

Despite the great advance of prior research in studying the discussion in issue reports and the characteristics of well-written reports, the characteristics of issue reports with raised questions remain unexplored. For example, even though an issue report is well-written, developers may still ask input from other team members to better understand the issue report. A high number of raised questions might also indicate that the solution to an issue report is not trivial (e.g., a high severity report), so that the developers have to carefully reach consensus before committing the final solution. In this paper, we conduct an in-depth analysis about the issue reports with raised questions. We aspire to provide earlier warnings (i.e., at the issue report filling time) to developers and users regarding issue reports that will likely receive questions, so they can provide the missing information while it is still fresh in their minds. Although not all questions may be avoided, warning that an issue report is more likely to receive questions may also lead users and developers to be more alert and respond to the questions more quickly.

To understand the most influential factors that can trigger questions in issue reports, we investigate 519,645 issue reports of three well known and long-lived systems (i.e.,

Linux, Firefox, and Eclipse). We observe that there is a considerable proportion (24.87% to 47.04%) of issue reports with raised questions that were reported for these three systems. Dealing with the raised questions in such a large amount of issue reports does have a serious threat to the efficiency of the issue resolving process. Hence, we are interested in understanding what questions are raised and how such raised questions impact the issue resolving process.

First, we investigate what types of questions are asked in order to understand the reasons behind such questions and the information sought by developers when asking them. Next, we investigate the impact that raising questions has on the issue resolving process. Finally, we study the feasibility of predicting whether questions will likely be raised in a given issue report at the time that a report is created. In particular, we address the following three research questions:

RQ1: What type of questions are asked by developers? To extract the topics of questions that developers raised on issue reports, we apply the Latent Dirichlet Allocation (LDA) technique on the extracted questions from issue reports. Among the three subject systems (Linux, Firefox, and Eclipse), there are four common categories of questions "Current status", "Reproduce steps", "Final resolution" and "Operating system". The majority of questions are different across the three subject systems, as the type of questions heavily depends on the characteristics of issues belonging to each system. In particular, Linux developers ask more questions regarding "Configuration" and "Driver"; Firefox developers raise more questions related to "Safe mode" and "Reproduce steps"; and Eclipse developers ask more questions about *Component*. Although our findings only highlight the types of questions in the three systems, our approach can be applied to other systems for developers to tailor a particular checklist for their issue reports.

RQ2: Are issue reports with raised questions different from issue reports without raised questions?

It is intuitive to think that questions raised in issue reports will delay the resolution of such reports. However, it is important to empirically investigate whether issue reports with raised questions take a considerable extra time to be resolved. For example, although an issue report with raised questions can be more time consuming, the extra time that is needed could be of a small significance on the average population of issue reports.

We use statistical tests and measures of effect-size to evaluate the difference between issue reports with raised questions and without raised questions through four perspectives: (i) the time elapsed of the entire resolving process (i.e., from when the issue reports is "touched"[1] by developers for the first time to when it gets resolved); (ii) the number of developers involved in the resolving process; (iii) the number of comments; and (iv) the number of reassignments.

Our results show that issue reports containing questions have a longer elapsed time and involve more developers. In addition, although Bhattacharya and Neamtiu (Bhattacharya and Neamtiu, 2011) found that reassignments are not always correlated with issue resolution time, we observe that issues with raised questions are more likely to

[1] "Touched" refers to the activity of developers on an issue report, which could be either posting a comment on the issue report or changing any field of the issue report.

be reassigned to other developers. Our results suggest that issue reports containing questions are significantly different from issue reports without questions (in terms of (i), (ii), (iii), and (iv)).

RQ3: Is the occurrence of questions predictable?

We collect the metrics that are available at the creation time of an issue report, and experiment them with three modeling techniques (i.e., random forest, logistic regression and Naïve Bayes) to predict whether developers will raise questions on issue reports. We find that the model built with the random forest performs the best. The random forest model achieves an AUC value of 0.78 in the Linux system, 0.65 in the Firefox system, and 0.70 in the Eclipse system. It is feasible to predict the occurrence questions in the issue reports of Linux, or even Eclipse. The prediction can give developers an instant warning on newly created issue reports, so that developers and issue reporters can clarify the possible missing information while the issue is still fresh in the mind of the issue reporter.

Our main contributions are summarized as follows:

- We perform an in-depth analysis on the raised questions from the issue reports, such as what are questions raised, and how the raised questions impact the issue tracking process.
- We demonstrate the feasibility of building a model to predict the likelihood of questions raised in issue reports when such issue reports are submitted.

Paper organization. We provide a motivating example to our study in Section 2. We describe our data processing steps in Section 3. The experiment setup and results of RQ1, RQ2, and RQ3 are presented and discussed in Sections 4, 5 and 6, respectively. Threats to validity are outlined in Section 7. We discuss the related work in Section 8, and conclude the paper in Section 9.

2 Motivation Example

Little quantitative analyses have been conducted to investigate the impact of raised questions in the issue resolution process. A preliminary investigation of our data reveals that questions raised in issue reports might be indicative of delays in the issue resolution process.

Fig 1 shows an example of questions raised in the *Linux-53*[2] report. The Linux-53 report describes a problem about consistent I/O errors in the Linux kernel regarding the CDROM drive. The issue report then received comments from other users stating that the same problem was experienced in other Linux kernel versions.

In Comment-7, a developer (Diego) asks how to reproduce the issue. Diego mentions that he cannot reproduce the issue anymore, which is why he asks if there is an explicit way to reproduce the reported issue. Next, In Comment-8 (two years later), another developer asks whether anyone is still observing the issue. In Comment-9, after two months, another user (Seth) deciphered how to reproduce the issue, under a particular configuration, hardware, and software environment. Seth also provided a work-around to avoid the problem by disabling the *Direct Memory Access* (DMA)

[2] https://bugzilla.kernel.org/show_bug.cgi?id=53

Diego Calleja 2003-07-28 18:12:37 UTC Comment 7 [reply] [–]

2.6.0-test2:
I remember seeing this error in the past but I can't reproduce it anymore,
can someone reproduce it?

Adrian Bunk 2005-07-02 07:23:52 UTC Comment 8 [reply] [–]

Is anyone still observing this with recent 2.6 kernels?

Seth Goodman 2005-09-24 11:42:25 UTC Comment 9 [reply] [–]

Yes, I have the same problem as the OP on Debian Sarge and Debian testing (etch)
with both 2.6.11 and 2.6.12 kernels. My system has the LTN485S CDROM and
FW82801AA controller on the motherboard. Disabling DMA for this drive using

hdparm -d0 /dev/hdc

fixes the problem, even though Windows was able to use DMA on this drive on the
same hardware. The symptom is inability to mount a CD. Because this prevents a
successful Linux install from CD media, IMHO, this bug should be considered high
severity. The initial PIO bootloader on an install CD is able to read the CD
and load a kernel, but on booting into that kernel, it is unable to mount the CD
and the install cannot complete beyond that point. I had to install from
floppies to get this system running.

Nishanth Aravamudan 2006-05-09 14:37:34 UTC Comment 11 [reply] [–]

Seth, is this still a problem in 2.6.16/2.6.17 -- I know it's a bit of a pain,
but if you can keep us updated as to the bug's existence with the major
releases, at least, that'd be great.

Raul, are you sure you meant to respond to this bug? Your error appears to be
related to a USB drive?

Thanks,
Nish

Seth Goodman 2006-07-24 09:49:33 UTC Comment 13 [reply] [–]

I've changed the CDROM drive to get around this problem. If you consider it
important, I can put the drive model causing the DMA errors back in. I am now
running only stable (Sarge) with the current 2.6 kernel for Sarge.

Olaf Kirch 2007-03-12 03:07:40 UTC Comment 14 [reply] [–]

This bug hasn't seen any activity in 6 months. Okay to close?

Olaf Kirch 2007-03-22 01:37:00 UTC Comment 15 [reply] [–]

No feedback - closing.

Fig. 1: Motivating example for our study.

of the drive. Finally, in Comment-11, another developer (Nishanth) asks whether Seth was still experiencing the issue—probably to know whether the issue was fixed. However, Seth responded to Nishanth only after two months, stating that the problem was not experienced anymore, but Seth was using a different hardware. Finally in Comment-14, after six months, the assignee of the issue report (Olaf) asks again if anyone was still facing the problem. By waiting for 10 days without any response, Olaf considered the issue to be fixed and closed it.

Had the reporter of the issue (or any other user) quickly provided feedback for developers in all of the opportunities (i.e., Comment-7, Comment-8, Comment-11,

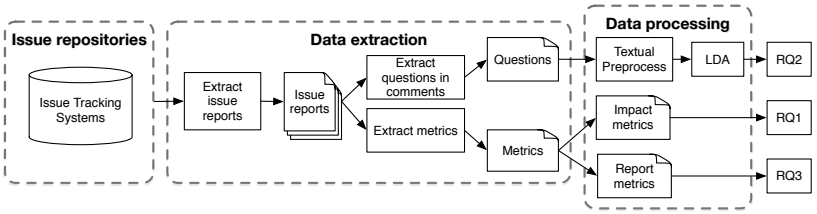

Fig. 2: Overview of our experiment

and Comment-14), the processing time of the Linux-53 issue report could had been much quicker.

Although prior research has studied the questions raised by developers and the characteristics of well-written issue reports (Bettenburg et al., 2008a; Breu et al., 2009), we are the first to quantitatively analyze the impact of questions raised by developers in the issue resolution process. We also build prediction models to capture issue reports that will likely have questions raised at their filling time. We analyze a much larger scale of issue reports than prior research (i.e., 519,645 issue reports). In particular, we study the Linux, Firefox, and Eclipse projects. The goal of our work is to improve the issue resolving process by providing early warnings for developers and reporters as to whether a newly reported issue will likely receive questions. By receiving such warnings, reporters should be more alert because their feedback will be necessary. As for developers, they might choose to prioritize an important issue report that will likely receive questions.

3 Data Processing

In this section, we present the dataset, and the steps to extract and process questions from issue reports. Fig 2 depicts the overview process of our data processing.

3.1 Subject Systems

We study issue reports from three popular ultra-large-scale and open-source systems, i.e., Linux[3], Firefox[4] and Eclipse[5]. Linux is known as a popular operating system. Firefox is a well-known web-browser. Eclipse has over 250 different open source projects, like the widely used integrated development environment (IDE) for software development, modeling tools, reporting tools, and much more. As the three systems are representative open source systems, studying issue reports in the three systems can reflect the practice in the open source community.

For each system, we collect all its issue reports from the first issue report until February 2016. For each issue report, we download all the properties and comments,

[3] https://bugzilla.kernel.org

[4] https://bugzilla.mozilla.org

[5] https://bugs.eclipse.org/bugs

Table 1: The number of issue reports in each studied ITS

ITS	Collection period	# of reports	# of resolved reports	# of reports with questions	# of reports without questions
Linux	2002 to 2016	27,100	23,134	10,882(47.04%)	12,252(52.96%)
Firefox	1999 to 2016	157,340	138,767	54,644(39.38%)	84,123(60.62%)
Eclipse	2001 to 2016	406,303	357,744	88,967(24.87%)	268,777(75.13%)
Total		590,743	519,645	154,493(29.73%)	365,152(70.27%)

```
Check :
http://www.alsa-project.org/alsa-doc/doc-php/template.php3?company=Creative+Labs&
card=Soundblaster+16&chip=sb16&module=sb16
```

Fig. 3: An illustrative example of a url link containing "?"

```
May 10 12:11:54 jtb [ 9215.990028]  [<ffffffff81058215>] ?
try_to_wake_up+0x1d1/0x1f6
May 10 12:11:54 jtb [ 9215.990034]  [<ffffffff8146b1a3>] ? printk+0x79/0x92
May 10 12:11:54 jtb [ 9215.990054]  [<ffffffffa004c46e>] ?
iwl_tx_agg_stop+0xda/0x212 [iwlcore]
May 10 12:11:54 jtb [ 9215.990059]  [<ffffffff8105f0c1>]
warn_slowpath_null+0x23/0x39
May 10 12:11:54 jtb [ 9215.990077]  [<ffffffffa0022763>]
___ieee80211_stop_tx_ba_session+0x69/0x94 [mac80211]
May 10 12:11:54 jtb [ 9215.990082]  [<ffffffff8146de8b>] ? _spin_lock_bh+0x20/0x4f
May 10 12:11:54 jtb [ 9215.990097]  [<ffffffffa00228e0>]
```

Fig. 4: An illustrative example of call trace message containing "?"

as well as the changes made to the attributes (e.g., severity, priority, #CC) in the entire history. As we aim to predict if questions can be raised from a newly created issue report (RQ3), we retrieve the initial value of the attributes of issue reports and issue reporters (details are discussed in Section 6.2) provided at the creation time of the issue reports.

To examine the impact of raising questions on the issue addressing process, we exclude the issue reports that are not resolved (i.e., issue reports still with the status of NEW, ASSIGNED and REOPENED). Table 1 shows the descriptive statistics of the issue reports that we collected from the issue tracking systems (ITSs) of the three subject systems.

3.2 Question Extraction

To investigate all raised questions across the issue resolving process, we go through all the comments of issue reports and extract the raised questions from the comments using regular expressions. Specifically, we consider a sentence ending with the question symbol "?" as a question. We only extract questions from comments which are posted by developers before the issue report is resolved. However, our extracted questions might contain noises. In summary, there might be three kinds of noise in our extracted questions:

```
Here is a snippet:
public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell (display);
        Button button = new Button(shell, SWT.PUSH);
        button.setText("button.");
        button.pack();
        button.setLocation(20, 20);
//      shell.setLayout(new GridLayout());
        display.addFilter(SWT.KeyDown, new Listener() {
                public void handleEvent(Event e) {
                        shell.setOrientation(shell.getOrientation() ==
SWT.RIGHT_TO_LEFT ? SWT.LEFT_TO_RIGHT : SWT.RIGHT_TO_LEFT);
                }
        });
        shell.open();
        while (!shell.isDisposed()) {
                if (!display.readAndDispatch()) display.sleep();
        }
        display.dispose();
}
```

Fig. 5: An illustrative example of a code snippet containing “?”

1. ***URL links (e.g., https://www.***./?)***. It is common that developers use a URL link in a comment to refer other issue reports or external pages to help address the issues. It is possible that a URL link contains the symbol of “?”. For example, Fig 3 shows a URL link appears in the comments of the report Linux-135. In this example, there exists the symbol of “?” in the URL link, and it is identified as a question. To exclude such kind of noise, we use a regular expression to identify all URL links (i.e., strings that start with "http://" or "ftp://") and exclude them.
2. ***Execution logs.*** It is common that developers paste execution logs in the comments to help discuss and investigate the reported issues. Fig 4 shows an example of a call trace embedded in a comment (i.e., in Comment-5 Linux-12595). We use a regular expression to identify the embedded call trace from the comments. The format of the call traces usually starts with a date and contains "address format" (e.g., [<ffffffffa002789f>]) before the symbol of “?”.
3. ***Code snippets.*** Code snippets embedded in a comment allow developers to discuss and investigate the potential reasons for generating the corresponding issue in the code snippet. However, code snippets can contain noises that are identified as questions. For example, Fig 5 displays an example of noise that is identified as a question in the code snippet embedded in Eclipse-29779 Comment-103. The symbol of “? :” is an operator in programming languages (i.e., C, C++, Java and so forth), and the symbol of “?” can exist in a code snippet. In this case, “?” is followed by an identifier and a symbol of “:” (e.g., return num>max?num:max). Hence, we exclude such noise using the regular expression to identify code snippets.

An issue reporter may submit additional comments that provide extra information to enrich the description of the issue report. As shown in Fig 6, the reporter posts two

Eclipse-Bug 1691:
Darin Wright [2001-10-10 22:17:42 EDT]:
Destription
When I get an error, I get a trace, it would be good to be able to select the Exception class, right click and add a breakpoint and re-run the execution up to the exception
...
Darin Wright [2002-02-26 14:52:26 EST]:
Comment1
Deferred
...
Darin Wright [2002-09-13 10:58:18 EDT]:
Comment2
Consider to 2.1
...
Darin Swanson [2003-10-06 13:47:58 EDT]:
Comment7
Just to check...but you meant to use the 1.4 support?
...

Fig. 6: An example of additional comment

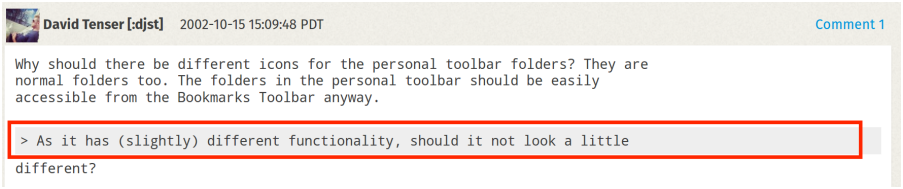

Fig. 7: An illustrative example of a replying comment

new comments (i.e., in Comment-1 and Comment-2) to enrich the information of an issue report. For example, in Comment-1 the reporter mentions that the resolution of the issue was deferred. In Comment-2, the reporter suggests that the issue report should be considered for version 2.1. Therefore, we start to extract questions from the first comment which is not posted by the issue reporter.

During the resolving process, developers might raise questions from the previous comments instead of the description of issue reports. To ensure the extracted questions are more related to the issue reports, we choose to use a conservative approach to filter out questions raised from the replying comments. The subject issue tracking systems support the feature of replying comments (i.e., start with the symbol of ">"), so we can identify the replying comments by the feature. As illustrated in Fig 7, a question (i.e., *Why should there be different icons for the personal toolbar folders?*) is raised in a comment due to an early comment (i.e., *As it has (slightly) different functionality, should it not look a little different?*). Hence, we exclude the extracted questions from the replying comment.

3.3 Question Processing

After collecting the raised questions from the comments, we apply general textual preprocessing steps, such as tokenizing text, removing stop words and stemming words. Tokenizing text is to obtain a sequence of strings that do not contain delimiters (e.g., white space and punctuation symbols). Stop words are the non-descriptive words like “a”, “is”, “was”, and “the” (Venkatesh et al., 2016). The stemming step aims to normalize the words to their ground forms (Tian et al., 2016). For instance, the stemmed version of “working” and “worked” is the same with “work”. We apply the Porter stemming algorithm as previous studies (Jones, 1997; Tian et al., 2016; Venkatesh et al., 2016).

4 RQ1: What type of questions are asked by developers?

4.1 Motivation

Studying the most frequently asked questions helps us to better understand the information that developers need when resolving issues. Prior work have studied 600 issue reports to catalog the types of questions that are asked in issue reports (Breu et al., 2010). Since our goal is to study the impact of questions raised on the issue resolution process, we expand this investigation using a larger scale of issue reports and additional systems (i.e.,Linux). To answer this question, we first present our approach to extract topics from questions and then discuss the extracted topics. Details are described in the subsections.

4.2 Experiment Setup

Differently from prior research Breu et al. (2010) that use a qualitative approach to analyze question types, we apply Latent Dirichlet Allocation (LDA) (Blei et al., 2003) to summarize the topics within raised questions. LDA is a statistical model that is widely used for topic extraction in the literature of software engineering. LDA uses a list of documents as its input. In our case, the input consists of a list of raised questions—if an issue report contains more than one question, all questions are treated as a single document. The output of LDA is the distribution probability of each extracted topic in each document.

We follow the standard natural language processing steps (see Section 3.3) to pre-process each document. Specifically, we remove stop words from each document and do the stemming to normalize the document.

LDA Parameter Setting

We run with 1,000 Gibbs sampling iterations by following the guideline from previous work (Griffiths and Steyvers, 2004), and set the number of keywords for each topic to be 20. The number of topics (we denote it by K) impacts the quality of the topics

extracted by LDA (Arun et al., 2010; Cao et al., 2009; Deveaud et al., 2014; Griffiths and Steyvers, 2004; Ponweiser, 2012). To find the optimal number of topics (we denote by K), we compute the following three metrics:

- *Arun*2010 (Arun et al., 2010) that is computed based on two matrices (i.e., Topic-Word and Document-Topics), which are generated from LDA. The lower value, the better.
- *CaoJuan*2009 (Cao et al., 2009) that calculates the cosine distance of topics. The minimum value of *CaoJuan*2009 indicates that the corresponding K is the optimal number of topics.
- *Griffiths*2004 (Griffiths and Steyvers, 2004; Ponweiser, 2012) that is computed based on an estimate multinomial distribution of K topics to words in the corpus. The maximum value of *Griffiths*2004 indicates that the corresponding K is the optimal number of topics.

We apply the *FindTopicsNumber* function from R package *ldatuning* by varying K from 2 to 500. As the *FindTopicsNumber* function takes a very long time to finish for a large dataset, we apply it on a statistically representative sample of 2,368 issue reports. We use the Sample Size Calculator[6] to find out our statistically representative sample (i.e., 2,368 issue reports) of our population (i.e., 154,493 issue reports) with a confidence level of 95% and a confidence interval of 2%. The Equations 1 and 2 demonstrate how to compute a representative sample size.[7]

$$ss = \frac{Z^2 \cdot (p) \cdot (1-p)}{c^2} \tag{1}$$

Where $Z = 1.96$ for a 95% confidence level, p is the chosen percentage (.5 should be used for the worst case scenario), and $c = 0.02$ is the confidence interval. The output value of Equation 1 (i.e., 2,401) should be adjusted to the entire population by using Equation 2.

$$adj(ss) = \frac{ss}{1 + \frac{ss-1}{pop}} \tag{2}$$

Where $ss = 2,401$, and $pop = 154,493$. The output of Equation 2 is 2,364. Our sample of issue reports was randomly selected from our 154,493 issue reports that contain raised questions. Since our sample is statistically representative, the distribution of topics in the 2,368 issue reports can represent the distribution of topics in the 154,493 issue reports. Therefore, the number of topics obtained from a statistically representative sample can represent the number of topics obtained from the 154,493 issue reports. The optimal number of topics suggested by the three metrics *Arun*2010, *CaoJuan*2009, and *Griffths*2004 are 107, 162, and 182, respectively. Accordingly, we set the number of topics as 150 in our experiment that is close to the three suggested Ks.

[6] https://www.surveysystem.com/sscalc.htm

[7] https://www.surveysystem.com/sample-size-formula.htm

Table 2: The distribution of topics in the issue reports for three subject systems. The topics in bold-face are the topics that are common in all studied systems.

Linux			Firefox			Eclipse		
Index	**Labeled Topics**	**Frequency**	**Index**	**Labeled Topics**	**Frequency**	**Index**	**Labeled Topics**	**Frequency**
1	Configuration	2,452(20.58%)	1	Safe mode	8,776(14.41%)	1	**Current status**	**4,750(4.68%)**
2	Driver	2,443(20.51%)	2	**Current status**	**4,005(6.58%)**	2	Component	3,299(3.25%)
3	Commits	1,867(15.67%)	3	**Reproduce steps**	**2,652(4.36%)**	3	**Reproduce steps**	**3,109(3.07%)**
4	**Current status**	**800(6.72%)**	4	Crash	1,457(2.39%)	4	**Final resolution**	**2,727(2.69%)**
5	**Final resolution**	**266(2.23%)**	5	Extension	1,246(2.05%)	5	Plugin	2,648(2.61%)
6	Sound & Speaker	159(1.33%)	6	Image feature	1,087(1.79%)	6	Package dependencies	1,783(1.76%)
7	Expired report	156(1.31%)	7	**Final resolution**	**1,055(1.73%)**	7	**Operating system**	**1,699(1.68%)**
8	**Operating system**	**123(1.03%)**	8	Cookies data	1,038(1.70%)	8	Version control	1,698(1.67%)
9	Concurrency	101(0.85%)	9	Browser	985(1.62%)	9	Jar dependencies	1,636(1.61%)
10	Email	96(0.81%)	10	Bookmark/History folder	922(1.51%)	10	Development environment	1,585(1.56%)
11	Suggestion	96(0.81%)	11	Page loading	871(1.43%)	11	Log information	1,582(1.56%)
12	Cache	89(0.75%)	12	User interface	869(1.43%)	12	Encoding	1,508(1.49%)
13	Closing report	72(0.60%)	13	Plugin	861(1.41%)	13	Requirement	1,364(1.35%)
14	Version Control	70(0.59%)	14	Tab groups	792(1.30%)	14	EMF	1,246(1.23%)
15	Memory	69(0.58%)	15	**Operating system**	**781(1.28%)**	15	Zip files	1,194(1.18%)
16	Frequency	65(0.55%)	16	Additional information	677(1.11%)	16	Suggestion	1,173(1.16%)
17	Level	65(0.55%)	17	Configuration	667(1.10%)	17	Workspace	1,158(1.14%)
18	Log information	62(0.52%)	18	Human oriented	630(1.03%)	18	Concurrency	1,128(1.11%)
19	**Reproduce steps**	**61(0.51%)**	19	Issue relation	610(1.00%)	19	JDT	1,094(1.08%)
20	Patch	59(0.50%)	20	Blocking issues	595(0.98%)	20	Solution	1,036(1.02%)
Topic [1]		9,171(76.99%)			30,576(50.21%)			37,417(36.90%)

[1] The total number of the issue reports that contain the questions associated with the 20 most frequent topics.

LDA Topic Labelling

Each LDA topic is represented by a vector of 20 words. To better understand the meaning of each topic, we asked four graduate students to manually label each topic based on its keywords (Barua et al., 2014). The four students independently assign labels. The four graduate students all study in computer science. One of them is a 3rd year Ph.D. student, the rest of them are master students. Moreover, three of them have at least four years of working experience. When different labels are assigned to the same topic, a discussion is conducted until a consensus is reached.

As the space of the paper is limited, we only present the 20 most frequent topics for each system in Table 2. For each topic, the frequency is computed as the number of issue reports that contain questions associated with the topic . We consider that an issue report (i.e., a document) is associated with topics, if the corresponding topics are assigned the highest score by LDA in the document.

4.3 Results

In Linux, three topics, i.e., *Configuration*, *Driver* and *Commits*, appear in more than a half (i.e., 56.76%) of all issue reports. The topic, *Configuration*, refers to questions raised by developers to clarify the hardware and/or software configuration of the machine of issue reporters. For instance, questions related to *Driver* are raised to clarify the detailed information of drivers for the reported issue. Questions associated with *Commits* are asked to find out which commit introduces the issue. We think that at least two types of questions (i.e., *Configuration* and *Driver*) can be reduced or even avoided by developing a tool to automatically collect the configuration and driver information for issue reporters.

In Firefox, the most frequently occurring topic is the *Safe mode*, which appears in 14.41% of all issue reports. Firefox has a large number of add-ons. In the safe mode of Firefox, all add-ons are disabled. When an issue is reported to Firefox, developers need to ensure that the issue is caused by bugs of Firefox not by bugs of add-ons. Therefore, developers usually ask issue reporters if Firefox is in the safe mode. To avoid such types of questions, we suggest issue reporters to reproduce the issue in the safe mode and explicitly mentions the usage of the safe mode in the description.

In Eclipse, there is no topic clearly occurring more frequently than others. As shown in Table 2, the 20 most frequent topics only cover 36.90% of the population. One reason may be that Eclipse project is embedded with multiple plug-in components. The reported issues can be caused by various reasons from different plug-in components and developers of different plug-in components have different focuses of the needed information in the issue resolving process. It may be difficult to provide a universal solution to reduce or avoid questions raised in the Eclipse issue reports.

As a summary, the majority of the 20 most popular topics vary across systems. The list of the 20 most frequent topics in each system is shown in Table 2. The coverage of the 20 most frequent topics also varies across systems. For instance, the 20 topics appear in 76.99%, 50.21%, and 36.90% of issue reports in Linux, Firefox, and Eclipse, respectively. The difference may be due to the characteristics of the three systems.

In Linux, developers deal with the kernel of an operating system to support different hardware/drivers and various configurations. In Firefox, developers maintain a widely used browser that is enhanced by many add-ons, and issue reports can be created against bugs of Firefox or its add-ons. In Eclipse, there are many subprojects that do not overlap too much with each other.

Among the 20 topics, there are only four common topics across systems. The four common topics are *Current status*, *Final resolution*, *Operating system*, and *Reproduce steps*. The topic *Current status* refers to questions that are raised by developers to check the current status of an issue report. The resolution of an issue report may be delayed for a long period (Zhang et al., 2012b), then developers lose track of the issue report. The topic *Final resolution* refers to questions that clarify the final decision made on issue reports, and it happens when developers cannot reach a consensus. This type of question is likely to be reduced or avoided if all involved developers decide to solve the issue only after the issue is fully discussed. Questions related to topics *Reproduce steps* and *Operating system* are generally essential to clarify the detailed steps and the environment to reproduce an issue. Issue reporters should try their best to provide as many details as possible on the reproduce steps and the environment.

From the results of the LDA modeling, the common topics of the questions raised from the issue reports are Current status, Final resolution, Operating system, and Reproduce steps among the three subject systems. However, the majority of raised questions are specific to each system.

5 RQ2: Are issue reports with raised questions different from issue reports without raised questions?

5.1 Motivation

There exist a considerable number of issue reports containing raised questions over the issue resolution process. As shown in Table 1, questions are raised in 47.04%, 39.38%, and 24.87% of issue reports in Linux, Firefox, and Eclipse, respectively. Given that we observe such high proportions of issue reports containing questions raised, we empirically study whether issue reports containing raised questions have different characteristics when compared to issue reports without questions. It is intuitive to think that raised questions increase the time to resolve issue reports. However, it is still important to study how large is the difference in magnitude of such a resolution time. Scientifically investigating intuitive claims in software engineering is important, since much of the common wisdom is quoted without scientific evidence (Bettenburg et al., 2008a).

On the other hand, if the time to resolve issue reports with raised questions is indeed significantly larger, more research and tools should be invested to address why such questions are asked in the first place (e.g., a reporter did not provide any feedback

to the developer regarding whether an issue was fixed). Then, the missing information should be provided from the start.

Our work quantitatively investigates a large scale of issue reports to understand the differences of issue reports with raised questions. To answer this question, we describe the metrics that we use to compare our issue reports and the null hypothesis. Finally, we discuss our findings on the differences between issue reports with raised questions and issue reports without questions. Details are described in the following subsections.

5.2 Experiment Setup

In our experiment, we quantitatively measure the characteristics of issue reports with raised questions in terms of four metrics.

- *The time elapsed* is the duration between the time when an issue report is responded by the developers (e.g., posting comments or modifying some fields of an issue report) and the time when the issue is resolved.
- *The number of developers* is the count of developers that are involved in the issue resolving process.
- *The number of comments* is the count of comments that are posted by the issue reporter or developers during the issue resolving process.
- *The number of assignments* counts the number of assignment/reassignments of developers of an issue report.

The smaller the value of the aforementioned metrics, the more efficient is the issue resolving process. Smaller values indicate that an issue is resolved quicker, involves fewer developers, derives fewer comments, and is assigned to the appropriate developer with fewer iterations.

Null Hypothesis

We divide issue reports into two groups for each system. The first group (as a control group) contains all issue reports that do not have questions raised during the issue resolving process. The second group (as a treatment group) contains all remaining issue reports that have questions. For each of the four aforementioned metrics, we test the following null hypothesis:

H_1^0*: There is no difference in the distribution of the values of the metric between the issue reports without and with raised questions.*

To test the null hypothesis, we apply the Mann-Whitney U test (Sheskin, 2003) with the 95% confidence level (i.e., p-value < 0.05). The Mann-Whitney U test is also known as the Wilcoxon rank sum test, which is a statistical method that does not have assumptions about the distribution of two assessed variables, i.e., the values of the assessed metric. If there is a statistically significant difference, i.e., p-value <0.05, we reject the null hypothesis and conclude that the distribution of the values of the corresponding metric is significantly different between the issue reports with and without questions.

Linux-Bug 915:
Greg Kroah-Hartman [2003-07-12 20:32:55 UTC]:
Can you attach your .config?
Roger Luethi [2003-07-14 11:50:20 UTC]:
Created *attachment-527*
Greg Kroah-Hartman [2003-07-15 12:42:23 UTC]:
Ok, I've duplicated this now, I'll work on it...

Linux-Bug 9147:
Erik Boritsch [2007-12-01 16:44:03 UTC]:
The bug is still there with CONFIG_ACPI_EC is not set. Any other ideas?
Sebastien Caille [2007-12-05 15:16:34 UTC]:
I tried to dump /proc/interrupts every 5 seconds. ... Something is definitely triggering the irq1 when acpi=on...
Daniele C. [2007-12-05 15:44:15 UTC]:
... maybe i8042 multiplexer is triggering a wrong IRQ.

Fig. 8: Examples of raised questions

After applying the Wilcoxon rank test, we apply the Cliff's Delta effect-size measure (Cliff, 1993). Cliff's delta is a non-parametric effect-size measure to verify the difference in magnitude of one distribution compared to another distribution. The higher the value of the Cliff's delta, the greater the difference of values between distributions. For example, if we observe a significant p value, but a $negligible$ effect-size, we consider that the observed difference between distributions is not significant. We use the thresholds provided by Romano et al. (2006) to interpret the effect-size measures. The thresholds are defined as follows: $delta < 0.147$ (*negligible*), $delta < 0.33$ (*small*), $delta < 0.474$ (*medium*), and $delta >= 0.474$ (*large*).

5.3 Results

The elapsed time of issue reports with raised questions is significantly larger than the elapsed time of issue reports without questions. Figure 9 shows the distribution of each metric in two groups, i.e., issue reports with questions (WQ) and issue reports with no questions (NQ). The result of Mann-Whitney U test shows that there exists a statistically significant difference between the issue reports with questions and without questions (i.e., p-value<0.05) in the elapsed time.

We reject the null hypothesis H_1^0 for this metric. Additionally, we observe that the group with questions has a longer elapsed time by comparing with the other group (i.e., control group). In fact, our effect-size measures indicate that the difference in the elapsed time between WQ and NQ is *medium* in all studied systems. This further indicates that the difference in the elapsed time is significant.

The result suggests that issue reports with questions may contain non-negligible waiting periods. There could be also an extra time to re-investigate the issue given

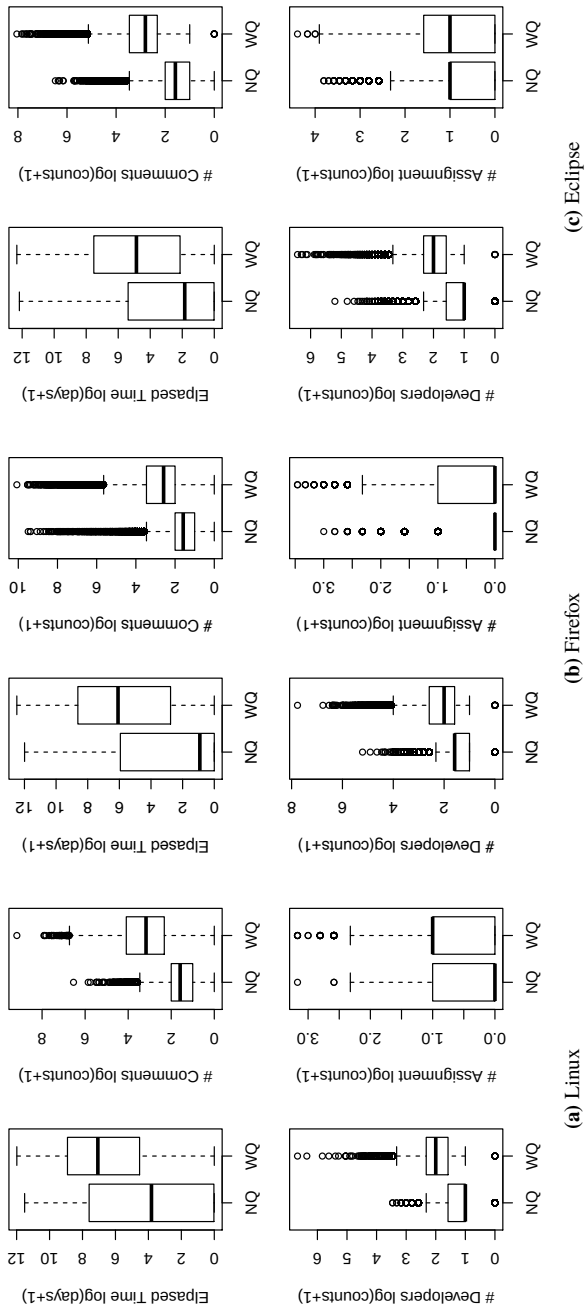

Fig. 9: The distribution of the metrics for issue reports without and with questions (NQ: Issues reports that have no questions; WQ: Issue reports With Questions). We show the value of metrics by log scale based on 2.

the new information provided in the answers. For instance, an issue report of Linux (i.e., *Linux-Bug 915* in Fig. 8) shows that the assignee of the issue report named Greg requested the configuration file from the reporter, and the reporter provided the requested file after about two days. Then, the assignee reproduced the issue with the newly attached configuration file after another day. If the configuration file was provided at the creation time, three days could had been saved in the resolving time of this issue report.

Issue reports with raised questions have more developers involved and more comments. The results of the Mann-Whitney U test show that the issue reports with questions have significantly more developers involved and comments (i.e., p-value $<$ 0.05) in the three subject projects. Indeed, we observe a $large$ effect-size difference for both number of developers and comments in all studied systems.

If a developer cannot address the issue report by her/himself, s/he may ask questions that involve other developers to answer. For instance, in *Linux-bug 9147* (Fig. 8), a developer encountered an unexpected issue that the bug still exists with a specific configuration, therefore he or she asked help from other developers who may be in charge of the particular module. The answer from the second developer further triggers the discussion with the third developer.

Issue reports with raised questions have a higher rate of reassignments. The results of the Mann-Whitney U test show that the issue reports with raised questions have more reassignments than the issue reports without raised questions (i.e., p-value $<$ 0.05). After analyzing the effect-sizes, we observe that the difference in reassignments is $small$ in the Linux and Eclipse systems. Nonetheless, we observe a $negligible$ effect-size in the Firefox project, which indicates that the number of reassignments is not significantly different between issue reports with questions and without questions in Firefox.

An assignee may raise questions if she or he does not have the time to fix the issue or cannot fix the issue (an example happened in Eclipse-bug 10025: *would you be able to fix this up?*), or another developer could be involved to make a decision on the issue report (an example happened in Eclipse-bug 123976: *Shouldn't this be resolved as invalid instead of fixed?*). In such cases, issue reports could be reassigned. The $negligible$ difference observed in Firefox, might indicate that Firefox has an effective triaging process. However, more analyses should be performed to confirm this speculation.

Issue reports with raised questions have a larger elapsed time and a higher number of developers, comments, and reassignments. These results suggest the importance of identifying why questions are raised in the first place and to check whether they could be avoided to speed up the issue resolving process.

6 RQ3: Is the occurrence of questions predictable?

6.1 Motivation

The results of RQ2 (see Section 5) show that issue reports with raised questions have a larger elapsed time, more developers involved, and a higher number of comments when compared to issue reports without questions. Given these observations, we investigate whether we can predict the occurrence of questions using only the information that is available at the creation time of the issue reports. This is important to provide both developers and reporters an early warning about whether a newly report issue will trigger questions. For example, such early warnings could inform reporters that they should be responsive to developers in case they want the issues to be resolved quickly.

To answer this question, we first describe the metrics that we use to build our prediction model. Then, we present our modeling techniques, performance measures and comparison. Finally, we discuss the predictive power of our influential metrics.

6.2 Experiment Setup

To address RQ3, we build prediction models. The dependent variable Y of our models (i.e., the output of our predictions) is whether an issue report will have raised questions. We code our dependent variable as $Y = 1$ if an issue report has raised questions and $Y = 0$ otherwise. We extract several metrics to be used as inputs for our models (i.e., independent variables). Finally, we validate our models using a 10-fold cross validation approach. We provide more details about each of these steps below.

1) Independent Variables. To better describe the features and the attributes of an issue report, we extract metrics from issue reports by four perspectives: the textual factors, the characteristic factors, the supportive factors and the historical factors. In total, we extract 13 metrics from issue reports.

a. Textual factors that are extracted directly from the textual data (e.g., the description and title of an issue report). We collect the following three metrics.

- *Length of the Title* is defined as the number of words contained in the title of an issue report. A longer title is more likely to provide sufficient information about the issue, and thus developers may raise fewer questions.
- *Length of the Description* is the number of words in the description field of the issue report. Similar to the title, a longer description is likely to provide more elaborate information about the issue.
- *Readability* is the Coleman-Liau index(CLI) (McCallum and Peterson, 1982) of the issue description, which has been applied in the evaluation of issue reports (Hooimeijer and Weimer, 2007). The CLI reveals how difficult to understand the text, and it is calculated as $CLI = 0.0588 * L - 0.296 * S - 15.8$, where L is the average number of the characters per 100 words, and S is the average number of sentences per 100 words. A lower readability is more likely to make developers confused and ask questions.

b. Characteristic factors that are extracted from the meta-data of an issue report (e.g., the importance and the complexity of a reported issue). We compute the following five metrics.

- *Is Regressive* is a boolean variable that indicates whether the issue report was reported and fixed before and reoccurs. Developers are more likely to raise more questions on the issues that have the regressive property.
- *Severity* describes the severity of an issue report, and ranges from 1 to 5 (i.e., "enhancement" to "blocking") in Linux, and from 1 to 7 (i.e., "enhancement" to "blocker") in both Firefox and Eclipse.
- *Priority* captures the priority of an issue report, and ranges from 1 (low priority) to 5 (high priority).
- *Is Blocking* is a boolean variable that indicates whether the issue must be addressed before addressing the issues that are listed in the "Blocks" field.
- *Is Dependent* is a boolean variable that indicates whether another issue must be addressed before addressing the reported issue.

c. Supportive factors that are extracted from the information used to assist developers to reproduce and resolve the reported issue. We calculate the following three metrics.

- *Has steps to reproduce* is a boolean variable that indicates whether an issue report describes the steps to reproduce the reported issue. We identify whether an issue report includes the *steps to reproduce* by searching the keywords "steps", "reproduce", and "steps to reproduce" in the description of the issue report. Developers are more likely to ask questions about how to reproduce the issue, if details of the reproduce steps are missing in the issue report.
- *The number of attachments* counts the attachments that include the patches and the testing case for the issue report.
- *The number of CCs* is the number of unique developers contained in the carbon-copy(CC) list of the issue report. We retrieve the history of activities in each issue report to use only the CCs that are provided at the issue reporting time. Hence, our model does not rely on the updated CCs that are later provided by developers. A developer listed in the CC field will get informed if there is a change in the issue report. More developers included in the CC list indicate a higher chance of interactions among developers in the early stages of an issue report.

d. Historical factors that are computed based on the history of the reporter that happened before the creation time. We compute the following two metrics.

- *Reputation of the reporter* is a float value to describe the reputation of the issue reporter. We compute the reputation of a reporter as the proportion of issue reports that are previously filed by the reporter and get fixed in the end (Guo et al., 2010b; Hooimeijer and Weimer, 2007): $reputation = \frac{|opened \cap fixed|}{|opened|+1}$. Note that 1 is added to the denominator in case a reporter did not report any issues in the past.
- *The rate of rejected issues previously reported.* Issue reports can be rejected as different resolutions, e.g., Worksforme, Duplicate, Invalid and so on. There are nine,

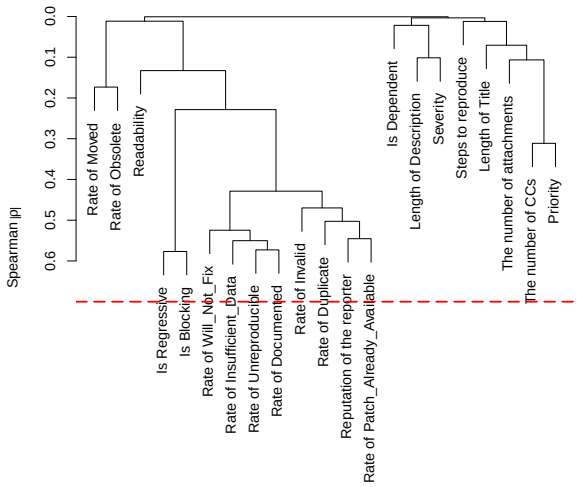

Fig. 10: The Spearman correlation analysis for Linux. The dotted red line indicates the threshold value of 0.7. We show only the correlation analysis for Linux for the sake of brevity, since the analyses for Firefox and Eclipse obtain a similar behaviour (i.e., none of the metrics are correlated).

seven and six types of resolutions for rejecting issue reports in Linux[8], Firefox[9] and Eclipse[10] respectively. To better describe the experience of developers, we use a list of float values to indicate the rate of different types of rejected issue reports previous reported by the reporter. We use T_i to denote the i_{th} type of rejected resolution (i.e., i = 1 to 9 in Linux, 1 to 7 in Firefox and 1 to 6 in Eclipse), and calculate $rate^{T_i}_{reporter}$ as $rate^{T_i}_{reporter} = \frac{|opened \cap T_i|}{|opened|+1}$.

2) Metric computation. The values of all aforementioned metrics are collected using only the information that is available at the creation time of issue reports. Therefore, we can apply these metrics to predict the chance of having questions raised at the creation time of an issue report. For the fields (e.g., severity, the number of CC) whose values may be changed during the issue resolving process, we trace back to the initial value that is filled in the creation time of an issue report.

3) Metric selection. The existing highly correlated factors can lead to overfitting of a model (Wikipedia, 2017). To analyze correlation, we calculate the Spearman rank correlation for our metrics to test whether there is any highly correlated pair of metrics. We apply the variable clustering technique (i.e., R function *varclus*) to see the correlation coefficient ρ. If the correlation coefficient of a pair of metrics is higher or equal to 0.7 (McIntosh et al., 2015; Rakha et al., 2015; Strobl et al., 2008), we

[8] https://bugzilla.kernel.org/query.cgi?format=advanced
[9] https://bugzilla.mozilla.org/query.cgi?query_format=advanced
[10] https://bugs.eclipse.org/bugs/query.cgi?format=advanced

consider the pair of metrics is highly correlated and pick only one of them. Fig. 10 depicts the Spearman rank correlation of the metrics in Linux as an example. The analyses for Firefox and Eclipse presented a similar behaviour, i.e., no correlation between metrics above 0.7. As there is no any pair of metrics that has the correlation coefficient value greater or equal than 0.7, we do not exclude any metrics in Linux, Firefox, and Eclipse.

Prediction Model

To build a better prediction model, we experiment with three different prediction models (Rahman and Devanbu, 2013), i.e., random forest, logistic regression and Naïve Bayes. Logistic regression is a generalized linear model which is a pretty well-behaved classification algorithm that can be trained as long as the features are expected to be roughly linear (Dayton, 1992; Hilbe, 2009). Unlike logistic regression, the random forest is an ensemble learning model and does not expect linear features or even features that interact linearly(Kamei et al., 2010; Liaw and Wiener, 2002). Naïve Bayes is a simple probabilistic classifier, which applies Bayes' theorem with the assumption of independence between every pair of features (Domingos and Pazzani, 1996; Rish, 2001).

Model Performance

To evaluate the performance of the prediction model, we use precision, recall, accuracy, and F-measure, which are explained below.

- Precision ($precision$) is defined as the proportion of the issue reports that are predicted as having questions and truly have questions (Davis and Goadrich, 2006; Powers, 2011): $precision = \frac{TP}{TP+FP}$.
- Recall ($recall$) measure the completeness of a prediction model. A model is considered as more complete if more issue reports with questions could be captured (Davis and Goadrich, 2006; Powers, 2011): $recall = \frac{TP}{TP+FN}$.
- F-Measure ($F\text{-}measure$) is the harmonic mean to describe both precision and recall measures in a single value (Rahman and Devanbu, 2013): $F\text{-}measure = 2 \times \frac{precision \times recall}{precision+recall}$.
- Accuracy ($accuracy$) calculates the percentages of the correct prediction (Rahman and Devanbu, 2013): $accuracy = \frac{TP+TN}{TP+FP+TN+FN}$.
- Area Under Curve (AUC) gives us an overview of the ability of the prediction model. AUC is the area under the curve plotting the true positive rate against the false positive rate. The AUC value could be ranged from 0.5 (worst, i.e., random guessing) to 1 (best, i.e., all targets could be captured by the prediction model).

Model Validation & Comparison

To obtain a reliable performance measure, we apply ten times of 10-fold cross-validation. In the 10-fold cross-validation, the data is equally divided into 10 parts. In each fold, the 9 parts are used to train the model and the remaining part is used to test

Table 3: The median value of performance measures of our models in the three studied systems. The values shown in the table refer to the prediction of the positive class, i.e., issue reports that received questions ($Y = 1$). Bold font highlights the performance of the best model in each system.

Model	System	Precision	Recall	F-Measure	Accuracy	AUC
Random forest	Linux	0.70	**0.70**	**0.70**	**0.72**	**0.78**
	Firefox	0.52	**0.57**	**0.54**	0.62	**0.65**
	Eclipse	**0.37**	**0.66**	**0.47**	0.63	**0.70**
Logistic regression	Linux	**0.71**	0.62	0.66	0.70	0.76
	Firefox	**0.53**	0.52	0.53	**0.63**	**0.65**
	Eclipse	**0.37**	0.55	0.44	0.65	0.66
Naïve Bayes	Linux	0.68	0.49	0.57	0.65	0.70
	Firefox	0.50	0.38	0.42	0.60	0.60
	Eclipse	0.36	0.39	0.38	**0.68**	0.63
Random guessing	Linux	0.47	0.50	0.48	0.50	0.50
	Firefox	0.39	0.50	0.44	0.50	0.50
	Eclipse	0.25	0.50	0.33	0.50	0.50

the model. To compare the performance of our selected prediction models, we apply the Scott-Knott effect size clustering (SK-ESD) (Tantithamthavorn et al., 2015) on the results of the ten times cross-validation. The Scott-Knott effect size clustering ranks the performance based on the effect size of their differences. We use the R package *ScottKnottESD* for the comparison.

6.3 Results

It is feasible to predict the occurrence of questions solely based on the information that is available at the creation time of the issue reports. Table 3 shows the precision, recall, F-Measure, accuracy, and AUC of our prediction models. For instance, the random forest model achieves the AUC values at 0.78 in Linux, 0.65 in Firefox, and 0.70 in Eclipse, significantly outperforming the random guessing with a large margin (i.e., **from 0.15 to 0.28**). The random forest model generally performs the best in our selected models in terms of F-measure and the AUC value. The result of the SK-ESD shows that the AUC value achieved by the random forest model is statistically significantly higher than the logistic regression model and Naïve Bayes model, except in Firefox where the logistic regression model achieves the similar performance than the random forest model (i.e., both with a median AUC of 0.65).

During the issue resolving process, additional information (e.g., discussions among developers) can be collected as developers progress. The additional information that can be collected at a later stage of the issue resolving process is likely to improve the performance of the model in predicting the raising questions. However, since our goal is to give developers an early warning when a new issue report is created, it is worth mentioning that our model is purposely built under a difficult setting (i.e., all metrics are collected using only the information available at the creation time, and any other information that can be collected at a later stage of the issue resolving process is not used). As a summary, we conclude that it is feasible to predict

Table 4: The five important metrics in the random forest models. The metrics are clustered and ranked using Scott-Knott effect size clustering.

ITS	Rank	Metric	Importance
Linux	1	Number of CCs	0.1048
	2	Priority	0.0151
	3	Readability	0.0114
	4	Number of attachments	0.0108
	5	Description length	0.0076
Firefox	1	Number of CCs	0.0419
	2	Severity	0.0076
	3	Reputation	0.0069
	4	Rate of DUPLICATE	0.0059
	5	Description length	0.0058
Eclipse	1	Number of CCs	0.0255
	1	Reputation	0.0255
	2	Description length	0.0173
	3	Rate of DUPLICATE	0.0151
	4	Rate of WORKSFORME	0.0114

the occurrence of questions for newly created issue reports, especially in Linux and Eclipse.

Influential Factors

To find the influential factors, we apply the permutation test technique (Strobl et al., 2008). Specifically, each metric is randomly permuted and a model is built with the permuted metric. We use the *importance* R function from the *RandomForest* package to conduct the permutation test. Next, we apply the Scott-Knott effect size clustering (SK-ESD) (Tantithamthavorn et al., 2015) to group the importance scores of the metrics. If there is no significant difference in the importance scores between metrics, those metrics are considered as having the same rank.

The number of carbon-copy(CC) is the most influential factor in all three subject systems. Table 4 shows the average importance value of all the top five influential metrics based on the results of Scott-Knott effect size clustering. We observe that the *number of CCs* is the most important metric in our three studied systems. A long CC list may indicate that the issue reporter intentionally involves more developers in the issue report. From one perspective, the involvement of more developers is likely to trigger questions. However, a short CC list may also trigger questions by developers who notice the issue report later. Figure 11 shows an example of this situation in the Linux-32 issue report. In Comment-2 (4 days after the issue was reported), the developer Luke-Jr adds himself in the CC list and asks whether someone else is already working on the issue. In Comment-3 (after one day), the developer Khoa mentions that James (another developer) could be willing to tackle the issue report based on his list of maintainers.

It would be useful if ITSs could provide suggestions for reporters about whom they should add in the CC list. Had Luke-JR, Khoa, and James been listed from the start, the developers could have saved 5 days. These CC-list suggestions could be based on the statistics of the ITSs or on data provided by developers themselves (e.g., the list of

Fig. 11: Additional question by a developer who noticed the issue report later (Linux-32 report).

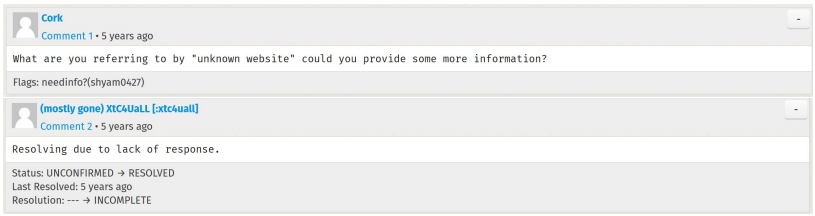

Fig. 12: The issue report Firefox-886686 was closed due to the lack of response from the reporter.

maintainers kept by Khoa in Comment-3 of Figure 11). In any case, reporters should provide a detailed description. This may not prevent all questions raised during the discussion, but at least can avoid questions raised against the issue description.

Moreover, each studied system obtains a different important metric in the second rank. The second most important metrics in Linux, Firefox, and Eclipse, are *priority*, *severity*, and *reputation*, respectively. The *priority* and *severity* metrics denote the importance level of a given issue report.[11] This level of importance is strongly associated with the likelihood of raising questions in issue reports (in the Linux and Firefox projects). If a reporter is aware that his/her reported issue has a high priority or severity, s/he should try the best to be responsive to the questions of developers. Figure 12 shows an example of a high priority and severity (i.e., P1 critical) issue report that was resolved as *INCOMPLETE* due to the lack of response from the reporter. In Comment-1, the developer (Cork) asked for clarification in the same day that the issue was reported. After two months, in Comment-2, the issue was closed due to the lack of response. If the reporter had provided feedback sooner, the report could have been closed much earlier.

On the other hand, *reputation* is the second most important metric in Eclipse. A reporter with a high *reputation* may know how to better write an issue report,

[11] https://www.mediawiki.org/wiki/Bugzilla/Fields#Importance

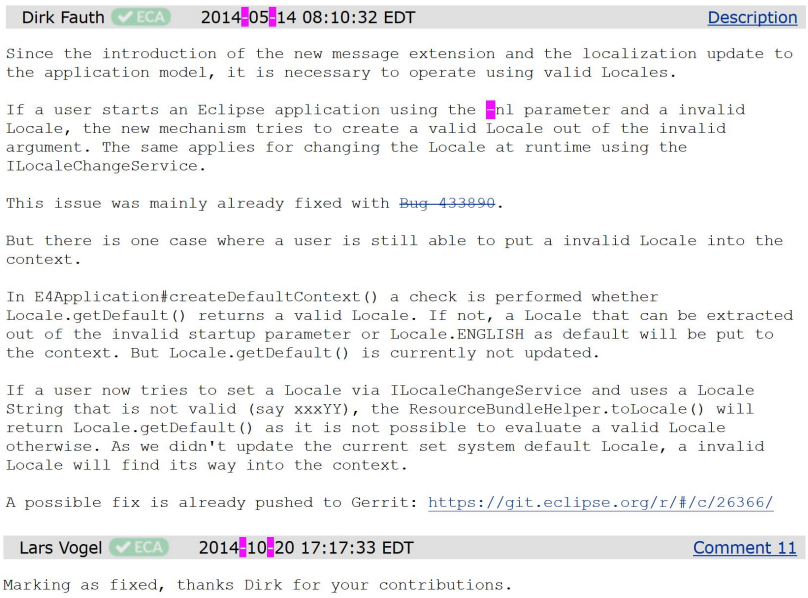

Fig. 13: The reporter of Eclipse-434846 has a high reputation of 0.94 in our data. He provided a contribution to the project when reporting his issue.

which implies that less questions are asked in such reports. Another explanation is that reporters with a high reputation may gain the trust of developers, which lead to less questions asked, whereas a lower reputation may lead developers to ask more questions. Issue reporters should actively work to build a good reputation within the project. For example, reporters should actively answer the questions of developers, participate in the discussions of other issue reports, and report high quality issue reports (Bettenburg et al., 2007). Another factor in play is whether the reporter can present a contribution to the project. Figure 13 shows an example of an issue report from Eclipse (Eclipse-434846 report) that was written by a high-reputable reporter (his reputation value is of 0.94 in our data). We observe that the reporter provided many details in the description of the issue report, while also providing a possible patch for the fix. In Comment-11, the developer Lars-Vogel explicitly thanked the reporter for his contributions. A similar dynamic was observed by Gousios et al. (Gousios et al., 2015) in which pull-requests are more likely to be accepted if they are submitted by contributors with a high reputation.

Finally, we observe that the *description length* is also a common important metric among the three studied systems. This result resonates with the intuition that the amount of detail provided in an issue report is likely to influence the number of questions raised in such a report. Reporters should write the description of their issues carefully and provide useful details—steps to reproduce, logs, error messages, operational system, hardware, and the reporter's attempts to overcome the issue (if applied).

It is feasible to predict if future questions will likely be raised by developers in issue reports. We recommend reporters to try their best to write CC lists that involve the appropriate developers, build a strong reputation within the project, provide a detailed description in their issue reports, and provide feedback as quickly as possible.

7 Threats to validity

In this section, we discuss the threats to the validity to our study, by following Yin's guidelines (Yin, 2013) of case study research.

External Validity concerns the threats to generalize our findings. Our three subject systems are representative open source systems that have a long history. As a result, our findings can reflect the current practice in open source community. Although our findings may not directly applicable to proprietary systems, our approach can be applied to any system with an issue tracking system to find the common types of questions, the impact of raising questions and the feasibility of predicting the occurrence of questions.

Internal Validity concerns the threats coming from the analysis methods and the selection of subject systems. The threat to our internal validity comes from the extraction of questions. Issue reports are free-form text, thus there exist different kinds of noise (i.e., link information, log information that contain question mark). To mitigate the noise, we manually examine the patterns from a sample set of issue reports. In addition, our text metrics may express different complexities despite their sizes, i.e., different people may describe the same problem using more or less words, or even more complex terms.

Another internal threat is that the extracted questions can be raised against a previously posted comment other than the description. We excluded the comments that are posted to explicitly reply a previous comment. However, it is worth adding more criteria to improve the accuracy of extracting questions.

Conclusion validity concerns the relationship between the treatment and the outcome. In our study, the elapsed time may be over-estimated, as the assignee might be working on other issues at the same time or is taking vacations for many days. Therefore, we also report three other metrics (i.e., the involved developers, the number of comments and the number of assignment) to measure the impact of raising questions on the issue resolving process. Our findings remain consistent across the four metrics. Nonetheless, it is helpful to obtain the ground truth by directly interviewing developers.

The raised questions may have positive effects. For example, the raised questions can make the issues resolved in a correct way, and cause fewer regressions or reopens. However, from a statistical comparison of our effort metrics between the control group (i.e., the issue reports do not have questions) and the treatment group (i.e., the issue reports containing questions), the issue reports containing questions generally have a larger elapsed time, number of developers, number of comments, and number of reassignments. Nevertheless, it is important to note that we cannot imply a causal

relationship between the increased values of, for example, elapsed time and the act of asking questions in issue reports. For example, issue reports with raised questions may be more associated to feature requests rather than bugs, which could take more time to complete. Our goal is to measure the characteristics of issue reports with raised questions. This is important to be aware that questions are likely to be raised, so that users and developers can provide more information at the issue reporting time or act more quickly when a question is asked.

8 Related Work

In this section, we discuss the research related to our study. Since we study an aspect of the issue resolution process (i.e., the questions raised by developers), we first discuss the research related to the issue resolution phases (i.e., triaging, resolution, and integration). Later, we group the related work into two categories. The first category consists of studies about developer discussions in issue reports and studies on the characteristics of well written reports.

8.1 Issue Resolution Process

In recent years, much research has been devoted to the issue resolution process (Anbalagan and Vouk, 2009; Anvik et al., 2006; Bhattacharya and Neamtiu, 2011; Giger et al., 2010; Guo et al., 2010a; Herraiz et al., 2008; Hooimeijer and Weimer, 2007; Kim and Whitehead, 2006; Marks et al., 2011; Panjer, 2007; Saha et al., 2014; Weiβet al., 2007; Zhang et al., 2012a, 2013). First, one must decide whether an issue is worth resolving and which developer should be responsible of an issue report Anvik et al. (2006). This process is mostly known as *issue triaging*. Hooimeijer and Weimer (2007) built models to classify the cost of triaging an issue report (e.g.,“cheap” or “expensive”). Kim and Whitehead (2006) studied the time required to resolve issues in the ArgoUML and PostgreSQL projects. Additional research has also been devoted to study the required time to resolve issue reports. For example, Weiβet al. (2007) and Zhang et al. (2013) estimated the time to resolve an issue based on the similarity between two issue reports. Guo et al. (2010a) used logistic regression to study the likelihood that an issue report will be resolved. Guo et al. (2010a) obtained a precision of 0.68 and a recall of 0.64.

A closed issue report may also be *re-opened* if the solution for such an issue is deemed as unsatisfactory (Shihab et al., 2010; Xia et al., 2015; Zimmermann et al., 2012). Shihab et al. (2010) studies the re-opened issues of the Eclipse project. The authors build prediction models based on several factors, such as the work habits (e.g., weekdays), the amount of time to fix an issue, and the experience of the issue assignee. Their prediction models achieve F-measures up to 0.71. Xia et al. (2015) further improve the models proposed by Shihab et al. (2010), achieving F-measures up to 0.86.

When resolving issue reports, developers may also find that some issue reports are very similar to other reports. Such similar issue reports are marked as *duplicate*

reports (Bettenburg et al., 2008b; Rakha et al., 2016, 2017; Runeson et al., 2007; Sun et al., 2010, 2011). Duplicate reports can be harmful, since developers will likely invest effort on them before noticing that they are duplicate. Rakha et al. (2016, 2017) investigate the needed effort to identify duplicate issue reports and propose an automatic approach to identify such duplicate reports.

Finally, recent studies have investigated the integration and deployment phases of resolved issues. Studies performed by da Costa et al. (2017a,b) investigate the required time to deliver resolved issues to end users. Morakot et al. (2015, 2017) studied the risk of delaying software releases posed by certain issue reports. Contrary to the aforementioned studies, our work focuses on the characteristics of issues reports with raised questions.

8.2 Developer discussion in issue reports

There are few studies that investigate the questions raised by developers. For instance, Breu et al. (Breu et al., 2009, 2010) quantitatively and qualitatively analyze the questions raised by developers in 600 issue reports. They find that the active and constant participation of reporters is an important factor that affects the progress of resolving issues. Their work focuses on understanding the questions themselves, while we aim to study the characteristics of issue reports with raised questions and the feasibility to predict the occurrence of these questions. We also study a much larger scale of issue reports (i.e., 519,645 issue reports).

Sillito et al. (Sillito et al., 2006) study the questions raised by programmers during the software evolution tasks. They categorize 44 kinds of questions asked by programmers, and the questions are divided into four types (i.e., finding initial focus points, building on the points, understanding subgraph, and questioning over groups of subgraphs). Herbsleb and Kwanna (Herbsleb and Kuwana, 1993) investigate questions that are asked during the designing stage. Project requirements are the most concerned topics in the questions asked by designers. Different from these studies, our collected questions are raised in various stages of software maintenance tasks, such as issue triaging, issue resolving, code reviewing, and testing.

Erdem et al. (Erdem et al., 1998) investigate the questions raised by users, and develop a task model based on the questions. The task model could be applied in generating the explanation to questions from users. Comparing to their work, we highlight the characteristics of issue reports that contain questions during the issue resolving process, which are overlooked by their work (Erdem et al., 1998).

8.3 Characteristics of well-written reports

One of the reasons for raising questions in issue reports is to ask information that could have already been provided by reporters. Bettenburg et al. . (Bettenburg et al., 2007, 2008a) conduct a survey on 466 developers and reporters of three systems (Apache, Eclipse and Mozilla). The authors investigate what are the characteristics of well-written reports. In addition, the authors propose CUEZILLA, a tool that recommends

how reporters may improve the quality of their issue reports on the spot, based on information mined from the Issue Tracking System. Differently from prior work, our goal is to study the characteristics of issue reports with raised questions instead of studying and recommending the characteristics of a well-written report. Additionally, we study 519,645 issue reports, 154,493 containing raised questions, to quantitatively investigate the differences of issue reports with raised questions on the issue resolution process. Finally, we propose models (Random Forest, Logistic Regression, and Naïve Bayes) to predict whether an issue report will likely raise questions by using metrics that can be collected during the filing time of an issue report. Although Bettenburg et al. (Bettenburg et al., 2008a) observe that developers give little importance to attributes such as severity and priority, our models show that these attributes are important in our Linux and Firefox models to predict the occurrence of questions.

Erfani Joorabchi et al. (Erfani Joorabchi et al., 2014) observe that 17% of their issue reports data consists of non-reproducible issue reports. The authors observe that such non-reproducible issue reports lead developers to spend a considerable time and effort on them. In our work, we observe that issue reports with raised questions take longer to be resolved. Herbold et al. (Herbold et al., 2011) indicate that the reproduction steps are the most important information in issue resolving, and introduce a non-intrusive GUI usage monitoring mechanism to support issue reproduction. Rohem et al. (Roehm et al., 2013) also provide a tool to monitor the interactions between users and their applications, which could help developers analyze the user interaction traces and derive steps to reproduce from the monitored interaction traces. We find that "Reproduce steps" is one of the common types of questions among our studied systems. Hence, we conjecture that the tools proposed by Herbold et al. (2011) and Roehm et al. (2013) may be applied to reduce the chances of raising questions to some extent.

9 Conclusion

When resolving issue reports, it is common for developers to ask questions to clarify the issue description. However, extra time is added to the issue resolving process, as developers need to wait for the response after raising questions. In this paper, we conduct an empirical study to understand the characteristics of issue reports with raised questions. We investigate three representative open source systems including Linux, Firefox and Eclipse. We observe that questions are raised in 47.04%, 39.38%, and 24.87% of all resolved issue reports in the Linux, Firefox, and Eclipse projects, respectively.

First, we apply the LDA to extract the topics of questions. We find that there are only four common topics of questions, including *Current Status*, *Final resolution*, *Operating system* and *Reproduce steps* in the three subject systems. In the three subject systems, the majority questions vary across systems. The topics identified by our approach can help developers understand the reasons behind the raised questions.

Second, we measure the characteristics of issue reports containing questions in terms of four measures: (i) the time elapsed for resolving an issue, (ii) the number of involved developers, (iii) the number of comments, and (iv) the number of assignments.

We observe that issue reports with raised questions have significantly greater values in all of the four studied metrics. Therefore, it is important for the development team to understand why many questions are raised and provide the information in advance whenever possible.

Finally, we investigate the feasibility of predicting the occurrence of questions at the creation time of an issue report. We collect 13 metrics that are available at the creation time and experiment them with three prediction modeling techniques. We find that the random forest model achieves the best performance in terms of F-measure and the AUC values. In particular, our model achieves the AUC value of 0.78, 0.65, and 0.70 in the Linux, Firefox, and Eclipse projects, respectively. Therefore, we conclude that it is feasible to give developers and reporters an early warning of the issue reports that are likely to raise questions. The issue reporters and developers should pay more attention to the issue reports with a high chance of having questions raised. More importantly, we recommend developers to provide a full checklist or design a tool to automatically check if necessary information is missing in a newly created issue report. Also, ITSs could provide a feature that warns the involved people when a question (not an usual comment) was asked in the issue report. Another important improvement would be to indicate the developers who should be in the CC list from the start.

In the future, we plan to work closely with developers to design a rule-based tool to highlight the missing information in a newly created issue report, and integrate our prediction model to the issue tracking system. Moreover, we are interested in quantifying how much unnecessary questions can be reduced, and how much the efficiency of the issue resolving process can be improved by removing unnecessary questions.

Acknowledgement

The authors would like to thank Wenhui Ji from Beihang University, Yongjian Yang and Pradeep Venkatesh, Mariam El Mezouar from the Software Reengineering Research Group at Queen's University for their valuable help on the manual labeling task for this paper.

References

P. Anbalagan and M. Vouk. On predicting the time taken to correct bug reports in open source projects. In *Proceedings of the 2009 IEEE International Conference on Software Maintenance*, ICSM '09, pages 523–526, Sept 2009.

John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

R Arun, Venkatasubramaniyan Suresh, CE Veni Madhavan, and MN Narasimha Murthy. On finding the natural number of topics with latent dirichlet allocation: Some observations. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 391–402. Springer, 2010.

Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

Olga Baysal, Michael W Godfrey, and Robin Cohen. A bug you like: A framework for automated assignment of bugs. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*, pages 297–298. IEEE, 2009.

Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 21–25. ACM, 2007.

Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318. ACM, 2008a.

Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Duplicate bug reports considered harmfulĚ really? In *Software maintenance, 2008. ICSM 2008. IEEE international conference on*, pages 337–345. IEEE, 2008b.

Pamela Bhattacharya and Iulian Neamtiu. Bug-fix time prediction models: Can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR)*, pages 207–210, 2011.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Frequently asked questions in bug reports. Technical report, University of Calgary, 2009.

Silvia Breu, Rahul Premraj, Jonathan Sillito, and Thomas Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 301–310. ACM, 2010.

Juan Cao, Tian Xia, Jintao Li, Yongdong Zhang, and Sheng Tang. A density-based method for adaptive lda model selection. *Neurocomputing*, 72(7):1775–1781, 2009.

Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. In *Psychological Bulletin*, volume 114, pages 494–509, 1993.

Daniel Alencar da Costa, Shane McIntosh, Uirá Kulesza, Ahmed E Hassan, and Surafel Lemma Abebe. An empirical study of the integration time of fixed issues. *Empirical Software Engineering Journal (EMSE)*, pages 1–50, 2017a.

Daniel Alencar da Costa, Shane McIntosh, Christoph Treude, Uirá Kulesza, and Ahmed E Hassan. The impact of rapid release cycles on the integration delay of fixed issues. *Empirical Software Engineering Journal (EMSE)*, pages 1–70, 2017b.

Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

C Mitchell Dayton. Logistic regression analysis. *Stat*, pages 474–574, 1992.

Romain Deveaud, Eric SanJuan, and Patrice Bellot. Accurate and effective latent concept modeling for ad hoc information retrieval. *Document numérique*, 17(1): 61–84, 2014.

Pedro Domingos and Michael Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classi er. In *Proc. 13th Intl. Conf. Machine Learning*, pages 105–112, 1996.

Ali Erdem, W Lewis Johnson, and Stacy Marsella. Task oriented software understanding. In *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*, pages 230–239. IEEE, 1998.

Mona Erfani Joorabchi, Mehdi Mirzaaghaei, and Ali Mesbah. Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 62–71. ACM, 2014.

Chiara Francalanci and Francesco Merlo. Empirical analysis of the bug fixing process in open source projects. In *IFIP International Conference on Open Source Systems*, pages 187–196. Springer, 2008.

Amir Hossein Ghapanchi and Aybuke Aurum. Measuring the effectiveness of the defect-fixing process in open source software projects. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–11. IEEE, 2011.

Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE)*, pages 52–56, 2010.

Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 358–368. IEEE Press, 2015.

Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 495–504, 2010a.

Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 495–504. IEEE, 2010b.

Steffen Herbold, Jens Grabowski, Stephan Waack, and Uwe Bünting. Improved bug reporting and reproduction through non-intrusive gui usage monitoring and automated replaying. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 232–241. IEEE, 2011.

James D Herbsleb and Eiji Kuwana. Preserving knowledge in design projects: What designers need to know. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 7–14. ACM, 1993.

Israel Herraiz, Daniel M. German, Jesus M. Gonzalez-Barahona, and Gregorio Robles. Towards a simplification of the bug report form in eclipse. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR)*, pages 145–148, 2008.

Joseph M Hilbe. *Logistic regression models*. CRC press, 2009.

Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 34–43. ACM, 2007.

Karen Sparck Jones. *Readings in information retrieval*. Morgan Kaufmann, 1997.

Yasutaka Kamei, Shinsuke Matsumoto, Akito Monden, Ken-ichi Matsumoto, Bram Adams, and Ahmed E Hassan. Revisiting common bug prediction findings using effort-aware models. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

Sunghun Kim and E. James Whitehead, Jr. How long did it take to fix bugs? In *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR)*, pages 173–174, 2006.

Andrew J Ko and Parmit K Chilana. Design, discussion, and dissent in open bug reports. In *Proceedings of the 2011 iConference*, pages 106–113. ACM, 2011.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Rafael Lotufo, Leonardo Passos, and Krzysztof Czarnecki. Towards improving bug tracking systems with game mechanisms. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 2–11. IEEE Press, 2012.

Lionel Marks, Ying Zou, and Ahmed E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 11:1–11:8, 2011.

Douglas R McCallum and James L Peterson. Computer-based readability indexes. In *Proceedings of the ACM'82 Conference*, pages 44–48. ACM, 1982.

Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, pages 1–44, 2015.

Choetkiertikul Morakot, Dam Hoa Khanh, Tran Truyen, and Ghose Aditya. Predicting delays in software projects using networked classification. In *30th International Conference on Automated Software Engineering (ASE)*, 2015.

Choetkiertikul Morakot, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Predicting the delay of issues with due dates in software projects. *Empirical Software Engineering Journal (EMSE)*, pages 1–41, 2017.

Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM, 2012.

Masao Ohira, Ahmed E Hassan, Naoya Osawa, and Ken-ichi Matsumoto. The impact of bug management patterns on bug fixing: A case study of eclipse projects. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 264–273. IEEE, 2012.

Lucas D Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the Fourth International Workshop on mining software repositories (MSR)*, page 29. IEEE Computer Society, 2007.

Martin Ponweiser. Latent dirichlet allocation in r. 2012.

David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 432–441. IEEE Press, 2013.

Mohamed Sami Rakha, Weiyi Shang, and Ahmed E Hassan. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*, pages 1–30, 2015.

Mohamed Sami Rakha, Weiyi Shang, and Ahmed E Hassan. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering*, 21(5): 1960–1989, 2016.

Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E Hassan. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *IEEE Transactions on Software Engineering*, 2017.

Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

Tobias Roehm, Nigar Gurbanova, Bernd Bruegge, Christophe Joubert, and Walid Maalej. Monitoring user interactions for supporting failure reproduction. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 73–82. IEEE, 2013.

Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In *Annual meeting of the Florida Association of Institutional Research*, 2006.

Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*, pages 499–510. IEEE Computer Society, 2007.

R.K. Saha, S. Khurshid, and D.E. Perry. An empirical study of long lived bugs. In *2014 Software Evolution Week - IEEE Conference onSoftware Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 144–153, Feb 2014.

David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 249–258. IEEE, 2010.

Jonathan Sillito, Gail C Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 23–34. ACM, 2006.

Carolin Strobl, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. Conditional variable importance for random forests. *BMC bioinformatics*, 9(1):1, 2008.

Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-*

Volume 1, pages 45–54. ACM, 2010.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 253–262. IEEE, 2011.

Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 2015.

Yuan Tian, Dinusha Wijedasa, David Lo, and Claire Le Gouesy. Learning to rank for bug report assignee recommendation. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

Pradeep K Venkatesh, Shaohua Wang, Feng Zhang, Ying Zou, and Ahmed E Hassan. What concerns do client developers have when using web apis? an empirical study of developer forums and stack overflow. 2016.

Cathrin Weiβ, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR)*, pages 1–, 2007.

Wikipedia. Multicollinearity — wikipedia, the free encyclopedia, 2017. URL `https://en.wikipedia.org/w/index.php?title=Multicollinearity&oldid=762815943`. [Online; accessed 6-February-2017].

Xin Xia, David Lo, Emad Shihab, Xinyu Wang, and Bo Zhou. Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering*, 22(1): 75–109, 2015.

Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.

Feng Zhang, F. Khomh, Ying Zou, and A.E. Hassan. An empirical study on factors impacting bug fixing time. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 225–234, Oct 2012a.

Feng Zhang, Foutse Khomh, Ying Zou, and Ahmed E. Hassan. An empirical study of the effect of file editing patterns on software quality. In *Proceedings of the 19th Working Conference on Reverse Engineering*, WCRE '12, pages 456–465, oct. 2012b.

Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, pages 1042–1051, 2013.

Thomas Zimmermann, Nachiappan Nagappan, Philip J Guo, and Brendan Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1074–1083. IEEE Press, 2012.